5.1 The X=USR(N) Statement

5.1 Adding simple USR routines

The USR(N) statement in BASIC is effectively a CALL to the routine whose starting address is given in BASIC workspace location USRLOC (1004/5H = 4100/1d). Thus,

```
      DOKE 4100, 3200    :REM Load USRLOC with 3200d = 0C80H
          .
          .
      A=USR(0)           :REM Equivalent to CALL (USRLOC)
```

is effectively CALL 0C80H.

Example 1.
The USR routine is to change the state of the tape drive led. The code to do this is to start at 0C80H (=3200d).
First the assembly language code to toggle the led:

```
                        ORG 0C80H
                MFLP    EQU 5FH
0C80 DF 5F              SCAL MFLP       ;Toggle TAPE LED
0C82 C9                 RET             ;Return to BASIC
```

This code may be entered into ram using the NAS-SYS M command or an assembler.(But see 5.1.4)

Next the BASIC instructions:

```
10   DOKE 4100,3200          :REM Point to USR routine
20   X=USR(0)                :REM Toggle led
30   FOR I=1 TO 999          :REM Delay so we can see the changes
40   NEXT I
50   GOTO 20                 :REM Do forever
```

Enter and RUN this program and observe that the tape led toggles on and off, ie. each time the USR instruction is reached the code at address 0C80 is executed.

5.1.2 Passing a USR result back to BASIC

Example 1 requires no transmission of data between BASIC and the USR routine. Many useful USR routines require that a single result be passed back from the USR routine to a BASIC variable. Thus, X=USR(0) may be required to place the result into BASIC variable X. (The result, X, must be in the range -32768 to +32767). To achieve this, the USR routine must place its result in register pair AB (sic), and then CALL the routine whose address is stored in locations E00D/E. This routine transfers the result in register pair AB to BASIC variable X. The USR routine should then RETurn to BASIC.

The USR routine should therefore have the structure:

```
      Code to place result
      in register pair AB.
      CALL   (E00D)      ;Call  the  routine   whose address
                           is in E00D/E.
      RET                ;Return
```

But a CALL (nn) istruction does not exist in the Z80 instruction set! However, the structure may be:

```
      Code to place result
        in register pair AB.
      LD HL,(0E00DH)
      JP (HL)
```

This causes a jump (rather than a CALL) to the location whose address is in location E00D. Since the code at this location ends with a RET instruction, it is that instruction which returns the USR routine to BASIC. This is illustrated in the following example.

Example 2.
The USR routine is to scan the keyboard once and return either the ASCII code for the key pressed or 00 if no key is pressed. (This is the INKEY statement available in some BASIC dialects.) First the assembly code to scan the keyboard once and place the result in the BASIC variable.

Listing of INKEY

```
                    0010 ;INKEY FOR BASIC
                    0020 ;SCAN KEYBOARD ONCE
                    0030 ;RETURN ASCII CODE OR 00 AS X.
                    0040 ;
                    0050 ;DOKE 4100,3200 TO USE
                    0060 ;THEN X=USR(0).
                    0070 ;
0C80                0080        ORG  0C80H
0C80 0061           0090 KBD    EQU  61H
                    0100 ;
0C80 DF61           0110        SCAL KBD
0C82 3801           0120        JR   C,CHAR
0C84 AF             0130        XOR  A
0C85 47             0140 CHAR   LD   B,A
0C86 AF             0150        XOR  A
0C87 2A0DE0         0160        LD   HL,(0E00DH)
0C8A E9             0170        JP   (HL)
                     180
                     190
```

Scan kbd once.
If key pressed, jump
else 00.
Save character in
reg pair AB.
Jump to routine
to put result into variable X
and return to BASIC.

As before, this code may be entered as hexadecimal code using the NAS-SYS M command, by an assembler, or the program in 5.1.4.

Next the BASIC code:

```
10 REM TEST INKEY
20 DOKE 4100,3200
30 X=USR(0)        :REM INKEY
40 PRINT X;
50 GOTO 30
```

Enter and RUN the program and observe that the PRINTed numbers are the decimal values of the ASCII codes of the keys pressed.

Another useful command in some BASIC dialects is GET, which waits for a single key to be pressed on the keyboard and returns with the ASCII code of that key. This is given below.

Example 3.
The USR routine is to blink the cursor until an input is received from one of the devices in the input table. The routine is to return with the BASIC variable having the ASCII code for the received character.
First the assembly language code to get an input character:

```
                0010 ;GET FOR BASIC
                0020 ;SCAN INPUT DEVICES, BLINK CURSOR
                0030 ;UNTIL INPUT DETECTED, THEN RETURN
                0040 ;WITH ASCII CODE IN BASIC VARIABLE.
                0050 ;
                0060 ;TO USE, DOKE 4100,3200
                0070 ;
0C80            0080        ORG   0C80H
0C80 007B       0090 BLINK  EQU   7BH
                0100 ;
0C80 DF7B       0110        SCAL BLINK
                0120 ;MOVE INPUT INTO BASIC VARIABLE.
0C82 47         0130        LD    B,A
0C83 AF         0140        XOR   A
0C84 2A0DE0     0150        LD    HL,(0E00DH)
0C87 E9         0160        JP    (HL)
```

Next the BASIC instructions:

```
10 REM TEST GET
20 DOKE 4100,3200
30 X=USR(0)    :REM GET
40 PRINT X;
50 GOTO 30
```

Observe that X is given the decimal value of the ASCII code for the received character.

5.1.3 Transmitting an Argument to the USR Routine

The USR(A) instruction may transmit the variable A to the USR routine. (Whatever the value of A, it is truncated by BASIC to an

integer and, if it is not in the range -32768 to +32767, an FC error will be displayed.)
The USR routine extracts the value of the argument A by calling the subroutine whose address is in location E00B/C. This routine transfers the BASIC variable A to register pair DE.

Example 4.
The USR routine is to toggle the TAPE LED a number of times, the number being determined by input to the BASIC program.
First the assembly language code to toggle the led a number of times:

```
0C80            0010        ORG   0C80H
0C80 005F       0020 MFLP   EQU   5FH
0C80 005D       0030 TDEL   EQU   5DH
                0040 ;Get argument into DE
0C80 2A0BE0     0050        LD    HL,(0E00BH)    ;Effectively
0C83 11880C     0060        LD    DE,ARG         ;a CALL (E00B)
0C86 D5         0070        PUSH  DE             ;with return to
0C87 E9         0080        JP    (HL)           ;ARG, argument in
0C88 4B         0090 ARG    LD    C,E            ;reg pair DE.
0C89 DF5F       0100 T1     SCAL  MFLP           ;Toggle led.
0C8B DF5D       0110        SCAL  TDEL           ;Delay.
0C8D 0D         0120        DEC   C              ;Again
0C8E 20F9       0130        JR    NZ,T1
0C90 C9         0140        RET
```

Note that the number of times the led toggles is determined by the lower byte of the double byte argument transmitted from BASIC. The BASIC argument is thus assumed to be in the range 1 to 256.
Now the BASIC code:

```
10 REM Toggle led N times
20 DOKE 4100,3200
30 INPUT "N";N
40 IF (N<1)OR(N>256) THEN 30
50 X=USR(N)
60 GOTO 30
```

5.1.4 Loading USR Code from BASIC

The USR code once in memory may be stored on cassette tape using the NAS-SYS W command and reloaded using the R command before activating BASIC.
However, it is more convenient to load a BASIC program with a USR routine in the same manner as any other program. This can be achieved by making the BASIC program itself load the machine code for the USR routine: the program begins with instructions to load the USR routine using DOKEs from a DATA statement. The DATA statement must give successive pairs of the machine code code as two byte signed decimal integers.

For Example 1, machine code DF,5F,C9 must be loaded at consecutive locations from 0C80:

```
          write as  DF 5F
                    C9 00
```

then reverse each pair:

```
                    5F DF
                    00 C9
```

and convert each pair to decimal:

```
                    5FDF   =  24543
                    00C9   =    201.
```

These are the decimal values to be DOKEd.

The BASIC program thus becomes:

```
 10 FOR I=3200 TO 3202 STEP 2
 20 READ A
 30 DOKE I,A
 40 NEXT
 50 DOKE 4100,3200
 60 X=USR(0)
 70 FOR I=1 TO 999
 80 NEXT
 90 GOTO 60
100 DATA 24543,201
```

For Example 2, the machine code from location 0C80 is:

```
DF 61
38 01
AF 47
AF 2A
0D E0
E9 00
```

Reverse each pair and convert to decimal:

```
61DF   =   25055
0138   =     312
47AF   =   18351
2AAF   =   10927
E00D   =   57357 , -65536 = -8179
00E9   =     233
```

Note that if the decimal value is greater than 32767, then 65536 must be subtracted to bring the number into the range of 16 bit twos complement numbers.

The BASIC program is thus:

```
 10 FOR I=3200 TO 3210 STEP 2
 20 READ A
 30 DOKE I,A
 40 NEXT
```

```
50 DOKE 4100,3200
60 X=USR(0)
70 PRINT X;" ";
80 GOTO 60
90 DATA 25055,312,18351,10927,-8179,233
```

The conversion from hexadecimal to decimal is tedious and error prone. The following BASIC program automatically generates the first lines of a BASIC program which makes use of assembly code routines.

```
80 REM MCLOAD
90 S=3200:F=3230
100 CLS
110 PRINT"10 FOR I=";S;"TO";F;"STEP 2"
120 PRINT"20 READ A"
130 PRINT"30 DOKE I,A"
140 PRINT"40 NEXT"
150 PRINT"50 DOKE 4100,";S
160 L=60
170 B=S
180 PRINT L;" DATA ";
190 FOR A=B TO B+6 STEP 2
200 PRINT DEEK(A);",";
210 NEXT
220 PRINT CHR$(8)
230 IF B+6>=F THEN 300
240 B=B+8
250 L=L+10
260 GOTO 180
300 NEW
310 END
```

To use the program, first enter the machine code for the USR routine into ram. Convert the first and last addresses to decimal:these are S and F respectively in line 90. (F-S must be odd.) Then load and RUN the BASIC program MCLOAD. It will print the BASIC lines required to load the USR routine. Move the cursor to the beginning of the first line and press the Enter key repeatedly to enter each line into the users BASIC program. Finally, LIST the program and enter the remainder of the required users program. When reloaded from tape, the program will automatically load the required USR code.

As an example assume that the multiple USR(N) function of the following section is to be incorporated into a BASIC program and that the machine code for USR(N) has been loaded into locations 0C80H to 0CC1H, ie. 3200 to 3265d.
Enter BASIC and load MCLOAD with line 90 changed to:

```
90 S=3200: F=3265
```

Then RUN MCLOAD and observe that the following is displayed:

```
10 FOR I=3200 TO 3265
20 READ A
30 DOKE I,A
40 NEXT
```

```
50 DATA 2858,4576,3208,-5675
```

followed by other DATA statements.
(In this particular example there are so many DATA statements that line 10 is scrolled off the screen, so it must be replaced.)
Now move the cursor to the beginning of the first line on the screen. Press Enter repeatedly and then LIST. This is the part of the program which loads the multiple USR(N) function.
To test, add the following lines:

```
150 INPUT"I";I
160 X=USR(I)
170 PRINT"USR";I,X
180 PRINT
190 GOTO 150
```

and SAVE the program. When this program is subsequently LOADed, USR(1) is the BASIC GET command, USR(2) is the BASIC INKEY command, and other functions return an Error message.

5.1.5 Multiple Function USR

Often more than one USR function is required in a program, particularly if it is a games program which must manipulate the video display rapidly. The following program shows how up to eight functions may be incorporated.

In the instruction X=USR(N), N is used to select the required routine. At line 110 the argument N is extracted and placed in register pair DE. Lines 160-180 force N to be in the range 0 to 7. It is then multiplied by two and added to the base address of the jump table, JTAB. Line 230 causes a jump into JTAB at the required position.
The jump table contains the jumps to each of the eight possible routines. Only USR(1) and USR(2) are actually used, the other routines simply write 'Error' to the video display. To add a routine, say USR(3), replace the line U3 JR ERROR with the required code and reassemble.

Listing of general USR function.

```
                    0010 ;GENERAL USR(N) FUNCTION
                    0020 ;N IS 0 TO 7
                    0030 ;TO USE DOKE 4100,3200
                    0040 ;THEN X=USR(N).
                    0050 ;
0C80                0060        ORG   0C80H
0C80 007B           0070 BLINK  EQU   7BH
0C80 006B           0080 ERRM   EQU   6BH
0C80 0061           0090 KBD    EQU   61H
                    0100 ;
                    0110 ;GET ARGUMENT INTO DE.
0C80 2A0BE0         0120        LD    HL,(0E00BH)
0C83 11880C         0130        LD    DE,ARG
0C86 D5             0140        PUSH  DE
0C87 E9             0150        JP    (HL)
```

```
0C88 1600      0160 ARG    LD   D,0
0C8A 7B        0170        LD   A,E
0C8B E607      0180        AND  7
0C8D 87        0190        ADD  A,A
0C8E 5F        0200        LD   E,A
0C8F 21940C    0210        LD   HL,JTAB
0C92 19        0220        ADD  HL,DE
0C93 E9        0230        JP   (HL)
0C94 180E      0240 JTAB   JR   U0
0C96 180E      0250        JR   U1
0C98 1814      0260        JR   U2
0C9A 1819      0270        JR   U3
0C9C 1819      0280        JR   U4
0C9E 1819      0290        JR   U5
0CA0 1819      0300        JR   U6
0CA2 1819      0310        JR   U7
               0320 ;
               0330 ;USR(0)
0CA4 1819      0340 U0     JR   ERROR
               0350 ;USR(1) IS GET
0CA6 DF7B      0360 U1     SCAL BLINK
0CA8 47        0370 RTA    LD   B,A
0CA9 AF        0380        XOR  A
0CAA 2A0DE0    0390        LD   HL,(0E00DH)
0CAD E9        0400        JP   (HL)
               0410 ;
               0420 ;USR(2) IS INKEY
0CAE DF61      0430 U2     SCAL KBD
0CB0 38F6                  JR   C,RTA
0CB2 AF                    XOR  A
0CB3 18F3                  JR   RTA
0CB5 1808      0470 U3     JR   ERROR
0CB7 1806      0480 U4     JR   ERROR
0CB9 1804      0490 U5     JR   ERROR
0CBB 1802      0500 U6     JR   ERROR
0CBD 1800      0510 U7     JR   ERROR
               0520 ;
0CBF DF6B      0530 ERROR  SCAL ERRM
0CC1 C9        0540        RET
```

If arguments are required to be transmitted from the BASIC program to the USR routine, they may be POKEd or DOKEd into free ram locations and then picked up by the USR routine. Similarly, if the USR routine is to transmit arguments back to BASIC, the USR routine should store them in free ram from where they can be picked up in BASIC using PEEK or DEEK.

5.2 Keyboard characteristics, K mode

The NAS-SYS Kx command allows the characteristics of the keyboard to be changed. The characteristics are determined by the contents of workspace loaction $KOPT (0C27H=3111d). The effect of a Kx command may be achieved simply by the BASIC statement:

POKE 3111,x where x=0,1,4, or 5.
Refer to the K command in Chapter 2 for the required value of x.

5.3 Activating User-written I/O, U mode

A user-written output routine may be activated by including UOUT in the table of output device. The required BASIC statement is
DOKE 3187,1912 (1918 for NAS-SYS 1)
The address of the start of the routine must of course have been placed in workspace location $UOUT+1/2 (0C78/9H=3192/3d) when the routine was loaded.

Similarly, a user-written input routine may be activated by
DOKE 3189,1915 (1921 in NAS-SYS 1)
Again, the start address of the routine must be in $UIN+1/2 (0C7B/CH=3195/6d).
Refer to the U command, Chapter 2, and the UOUT routine, Chapter 3.

5.4 Activating the External Serial Device, X mode

The external serial output driver requires an argument in workspace location $XOPT (0C28H=3112d); refer to the X command in Chapter 2, and the XOUT routine in Chapter 3.
The required BASIC statements are
DOKE 3187,1911 (1917 for NAS-SYS 1)
POKE 3112, x
where x=0,1,16, or 17.

The external serial input driver also requires an argument in $XOPT; refer to the X command, Chapter 2, and the XKBD routine in Chapter 3.
The required BASIC statements are
DOKE 3189,1919 (1925 for NAS-SYS 1)
POKE 3112,x
where x=0,2,16,18,32,34,48,or 50.

When both the serial output and input are activated together, argument x must be the logical OR of the input and output options.

5.5 Deactivating the U and X modes, N mode

The U and X modes may be deactivated by the NAS-SYS command, N, as described in Chapter 2. In BASIC this becomes
DOKE 3187,1913 (1919 in NAS-SYS 1)
DOKE 3189,1916 (1922 in NAS-SYS 1)